CLAIMS

We claim:

1. A method of modifying compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising the steps of:

providing compiled code;

scanning said compiled code for candidate instructions for substitution; and

substituting randomly generated, functionally isomorphic code in place of said candidate instructions to generate a first code polymorph.

2. The method of claim **1**, further comprising the steps of:

providing functionally isomorphic code as a first generation polymorph for further layers of modification;

scanning said first generation polymorph for further candidate instructions for substitution; and

substituting randomly generated, functionally isomorphic code in place of said further candidate instructions to yield a second generation polymorph.

3. The iterative application of the method of claim **2**, wherein the polymorph produced in the preceding iteration is modified in the next iteration to produce at least a third generation polymorph.

4. The method of claim **1**, further comprising the steps of:

modifying a first copy of compiled code to generate a first polymorph; and

modifying a second copy of compiled code to generate a second polymorph; wherein

said first and said second polymorphs have different physical instruction code but are substantially functionally isomorphic.

5. The method of claim 1, wherein the compiled code is the program code of a self replicating application.

6. The method of claim 1, wherein the code polymorph is generated on a server and downloaded over a network connection to a client platform.

7. The method of claim 1, further comprising the step of inserting random benign instructions in the polymorphed instruction code.

8. The method of claim 1, wherein the modification of code to generate a functionally isomorphic code polymorph is accomplished with a stand-alone software application.

9. The method of claim 1, wherein the modification of code to generate a functionally isomorphic code polymorph is accomplished with a call to a code library.

10. A system for modifying the compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising:

means for providing compiled code;

means for scanning said compiled code for candidate instructions for substitution; and

means for substituting randomly generated, functionally isomorphic code in place of said candidate instructions to generate a code polymorph.

11. A system for securely executing a first compiled application used to modify the instruction code of a second compiled application while substantially preserving said second application's original operational and functional characteristics, comprising:

means for enciphering said first compiled application;

means for decrypting an instruction code of said first application;

means for executing said decrypted instruction code of said first application;

means for re-enciphering said decrypted instruction code of said first application prior to decryption and execution of the next line of instruction of said first application; and

means for moving through the enciphered first application repeating the steps of decryption, execution and re-enciphering of instruction code to execute said first application.

12. A system for modifying the compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising:

means for providing compiled code;

means for substituting random context instruction codes for original CPU instructions in said compiled code;

and means for correlating said random context instruction codes to said original CPU instructions in order to recover said original CPU instructions.

13. A system for modifying the compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising:

means for providing compiled code;

means for detecting original program execution jumps that require accurate header data to properly execute;

means for overwriting said header data to render said original execution jumps ineffective;

means for redirecting program execution at said jump points to a resolution subroutine; and

correlation means enabling said resolution subroutine to execute a program jump to an appropriate codebase with accurate header information.

14. In a system for executing the randomly modified code of an application while substantially preserving the application's original operational and functional characteristics, a sub-system for securing user input, comprising:

an operating system capable of receiving input;

means for entering input; and

means for intercepting and obscuring said input before said input is made available to an operating system process.

15. The system for executing randomly modified code according to claim **14**, further comprising:

said operating system having a hardware device controller layer capable of receiving input;

said means for entering input having hardware means for providing encrypted data to said hardware device controller layer;

an application capable of actuating said means for providing encrypted data to request provision of encrypted data traffic; and

said application having means for decrypting and obscuring said encrypted data traffic from an operating system process.

16. In a system for executing the randomly modified code of an application while substantially preserving the application's original operational and functional characteristics, a sub-system for securing user input, comprising:

means for graphically displaying a keypad;

means for initial random assignment of values to the keys of said keypad;

means for identifying a key to be selected;

means for randomly positioning said means for identifying a key;

means for selecting a particular key; and

means for randomly assigning values to the keys of said keypad after each key selection is made.

17. A method of securely executing the code of an application on a CPU, comprising the steps of:

providing a compiled application having lines of CPU instructions;

enciphering the compiled application;

decrypting an instruction code to be provided to the CPU for execution;

executing said decrypted instruction code;

re-enciphering said decrypted instruction code prior to decryption and execution of the

next line of instruction; and

moving through the enciphered application repeating the steps of decryption, execution and re-enciphering of instruction code to execute the application.

18. The method of securely executing the code of an application on a CPU according to claim **17**, wherein the method of enciphering the compiled application further comprises the steps of:

calculating the length of the instruction code to be enciphered;

XOR'ing the first byte of said instruction code to be enciphered using a key;

encrypting the XOR'ed byte of said instruction code; and

moving through the application code repeating the steps of calculating the code length, XOR'ing the first byte of said instruction, and encrypting the XOR'ed byte of said instruction to encipher the remainder of the compiled application.

19. The method of securely executing application code on a CPU according to claim **17**, wherein the method of re-enciphering the executed instruction code further comprises the steps of:

calculating the length of the instruction code to be enciphered;

XOR'ing the first byte of said instruction code to be enciphered using a key; and

encrypting the XOR'ed byte of said instruction code.

20. The method of securely executing the code of an application on a CPU according to either one of claims **18** or **19**, wherein the key used to XOR the first byte of said instruction code to be enciphered is generated with a random number generation function.

21. A method for modifying the compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising the steps of:

providing compiled code;

substituting random context instruction codes for original CPU instructions in said compiled code;

correlating said random context instruction codes to said original CPU instructions in order to recover said original CPU instructions.

22. The method of modifying the compiled code of an application according to claim **21**, wherein the substitution of random context instruction codes for original CPU instructions further comprises the steps of:

calculating the original instruction code length;

using a matchable data structure to convert said original instruction into a random context instruction; and

placing said random context instruction on the same line as the original instruction.

23. The method of modifying the compiled code of an application according to claim **21**, wherein the recovery of an original CPU instruction from a correlated random context instruction is accomplished with a matchable data structure selected from the group consisting of a look-up table, a database query, a symmetric correlation, an asymmetric correlation, a functional correlation, a parametric correlation, a one-way function, and any combination thereof.

24. A method for modifying the compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising the steps of:

providing compiled code;

detecting original program execution jumps that require accurate header data to properly execute;

overwriting said header data to render said original execution jumps ineffective;

redirecting program execution at said jump points; and

executing a program jump to an appropriate codebase with accurate header information.

25. The method of modifying the compiled code of an application according to claim **24**, wherein redirection is accomplished by correlating EIP data with API place holder data.

26. The method of modifying the compiled code of an application according to claim **25**, wherein said API place holder data is encrypted.

27. A method of securing user input in a system for executing the randomly modified code of an application while substantially preserving the application's original operational and functional characteristics, comprising:

providing an operating system capable of receiving input;

entering input; and

intercepting and obscuring said input before said input is made available to an operating system process.

28. The method of securing user input according to claim **27**, wherein the means for entering input is selected from the group consisting of a keyboard, a mouse, a light pen, a stylus, a modem, a network card, a joystick, a paddle, a game controller, a wireless transmitter, a portable

digital assistant, a telephone, a mobile phone, a pager, a keypad, a trackball, a camera, and any combination thereof.

29. The method of securing user input according to claim **27**, wherein the means for obscuring said input is accomplished by encrypting said input.

30. The method of securing user input according to claim **27**, further comprising:

providing a hardware device controller layer communicably connected to said operating system and capable of receiving input;

providing hardware means for providing encrypted data to said hardware device controller layer;

actuating said means for providing encrypted data to request provision of encrypted data traffic; and

application level decrypting of said encrypted data traffic from an operating system process.

31. The method of securing user input according to claim **30**, wherein said means for entering input is selected from the group consisting of a keyboard, a mouse, a light pen, a stylus, a modem, a network card, a joystick, a paddle, a game controller, a wireless transmitter, a portable digital assistant, a telephone, a mobile phone, a pager, a keypad, a trackball, a camera, and any combination thereof.

32. The method of securing user input according to claim **30**, wherein said hardware device controller layer comprises at least one device driver.

33. The method of securing user input according to claim **30**, wherein said hardware device controller layer comprises a hardware abstraction layer.

34. The method of securing user input according to claim **30**, wherein said hardware means for providing encrypted data comprises an integrated circuit.

35. The method of securing user input according to claim **30**, wherein said application is a polymorphed code variant.

36. A method of securing user input, comprising:

graphically displaying a keypad;

providing means for initial random assignment of values to the keys of said keypad;

providing means for identifying a key to be selected;

matching said means for identifying a key with a random keypad value;

providing means for selecting a particular keypad value to be entered; and

randomly assigning values to the keys of said keypad after each keypad selection is made.

37. The method of securing user input according to claim **36**, wherein said means for identifying a key to be selected comprises a highlighting cursor.

38. The method of securing user input according to claim **36**, wherein said matching of said means for identifying a key with a random keypad value comprises positioning a highlighting cursor in the vicinity of the random keypad value.

39. The method of securing user input according to claim **36**, wherein said means for selecting a particular keypad value to be entered comprises a keyboard cursor key selected from the group consisting of an up arrow, a down arrow, a right arrow and a left arrow.

40. The method of securing user input according to claim **36**, further comprising the step of clearing the keypad display each time a keypad selection is made.

41. The method of securing user input according to claim **36**, further comprising the step of providing means for dynamically obscuring the display of keypad values.

42. The method of securing user input according to claim **41**, wherein the means for dynamically obscuring the display of keypad values comprises displaying a keypad value for a finite time in response to selection of a keypad value.

43. A method of securely executing the code of an application that has been modified in accordance with the method of claim **1**, comprising the steps of:

generating a first CRC of at least one system file at the time of code polymorph execution initialization;

generating a second CRC of said system file(s) while said code polymorph is executing; and

comparing said first CRC and said second CRC to determine if code access attempts are being made.

44. The method of securely executing the modified code of an application according to claim **43**, further comprising the step of periodically regenerating a second CRC while said code polymorph is executing for comparison to said first CRC to determine if code access attempts are being made.

45. The method of securely executing the modified code of an application according to claim **43**, wherein said system file(s) comprise dynamic link libraries.

46. A system for modifying the compiled code of an executable application while substantially preserving the application's original operational and functional characteristics, comprising:

a host server including a processor for processing digital data, a memory coupled to said processor for storing digital data, an input digitizer coupled to the processor for inputting digital data, a polymorphic engine application stored in said memory and accessible by said processor for directing processing of digital data by said processor, and a display coupled to the processor;

said host server being communicably connectable to at least one remote client platform over a network; and

said host providing a randomly polymorphed version of said executable application for communication to said remote platform.

47. The system for modifying the compiled code of an application according to claim **46**, wherein said network connection is selected from the group consisting of a LAN, a WAN, a VPN, the internet, an extranet, an intranet, and any combination thereof.

48. A method of modifying compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising the steps of:

providing compiled code;

scanning said compiled code for candidate instructions for substitution;

substituting randomly generated, functionally isomorphic code in place of said candidate instructions to generate a code polymorph; and

encrypting at least a portion of the code polymorph using a randomly generated encryption algorithm.

49. The method of modifying compiled code according to claim **48**, wherein the randomly generated encryption algorithm is executed prior to encryption of the code polymorph.

50. The method of modifying compiled code according to claim **48**, further comprising iterative application of the random encryption algorithm to the code polymorph using the result from the previous encryption iteration as a key for subsequent iterations.

51. The method of modifying compiled code according to claim **48**, wherein the randomly generated encryption algorithm is functionally symmetric.

51. The method of modifying compiled code according to claim **48**, wherein the encrypted polymorph produced thereby is distributed as a software application for provisional use.

52. A method of modifying compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising the steps of:

providing compiled code;

scanning said compiled code for candidate instructions for substitution;

substituting randomly generated, functionally isomorphic code in place of said candidate instructions to generate a code polymorph;

encrypting at least a portion of said code polymorph in response to a source code encryption request;

distributing said code polymorph produced thereby as a software application for provisional use;

calculating a signature substantially specific to the operating environment of said provisional software application; and

comparing a key to said signature for authorizing decryption and execution of said encrypted portion of said code polymorph.

53. A method of securely executing compiled code of an application while substantially preserving the application's original operational and functional characteristics, comprising the steps of:

providing a platform having a CPU, said CPU having a program stack;

providing compiled code;

dynamically replacing at least a first call in said compiled code with a second call which is written substantially directly to said program stack; and

removing said first call from said compiled code.

54. A method of securely executing compiled code of an application according to claim 53, wherein either said first call or said second call or both first and second call is encrypted.

55. A method of securely executing compiled code of an application according to claim 53, wherein said first call is an API call.

56. A system for protecting the compiled code of an executable application while substantially preserving the application's original operational and functional characteristics, comprising:

means for providing compiled code;

means for scanning said compiled code for candidate instructions for substitution;

means for substituting randomly generated, functionally isomorphic code in place of said candidate instructions to generate a code polymorph;

means for enciphering a first application;

means for decrypting an instruction code of said first application;

means for executing said decrypted instruction code of said first application;

means for re-enciphering said decrypted instruction code of said first application prior to decryption and execution of the next line of instruction of said first application;

means for moving through the enciphered first application repeating the steps of decryption and re-enciphering of instruction code to execute said first application;

means for substituting random context instruction codes for original CPU instructions in said compiled code;

means for correlating said random context instruction codes to said original CPU instructions in order to recover said original CPU instructions;

means for detecting original program execution jumps that require accurate header data to properly execute;

means for overwriting said header data to render said original execution jumps ineffective;

means for redirecting program execution at said jump points to a resolution subroutine;

correlation means enabling said resolution subroutine to execute a program jump to an appropriate codebase with accurate header information;

an operating system capable of receiving input;

means for entering input;

means for intercepting and obscuring said input before said input is made available to an operating system process;

said operating system having a hardware device controller layer capable of receiving input;

said means for entering input having hardware means for providing encrypted data to said hardware device controller layer;

an application capable of actuating said means for providing encrypted data to request provision of encrypted data traffic;

said application having means for decrypting and obscuring said encrypted data traffic from an operating system process;

means for graphically displaying a keypad;

means for initial random assignment of values to the keys of said keypad;

means for identifying a key to be selected;

means for randomly positioning said means for identifying a key;

means for selecting a particular key;

means for randomly assigning values to the keys of said keypad after each key selection is made;

means for generating a first CRC of at least one system file at the time of code polymorph execution initialization;

means for generating a second CRC of said system file(s) while said code polymorph is executing; and

means for comparing said first CRC and said second CRC to determine if code hacking attempts are being made.